Instruction Set Simulator generic API

This new API is an evolution of the ISS API. New features are:

- Use structs rather than arguments, this eases API evolution (which can be acomplished through carefully choosed default values);
- support for CPU modes;
- support for generic Memory management unit.

Structures and values

Enumerated values

Execution mode for any Instruction/Data access, checked by mode-enabled caches

```
enum ExecMode {
    MODE_HYPER,
    MODE_KERNEL,
    MODE_USER,
};
```

Operation type on Data cache access

```
enum DataOperationType {
    DATA_READ,
    DATA_WRITE,
    DATA_LL,
    DATA_SC,
    XTN_WRITE,
    XTN_READ,
};
```

When operation is XTN_READ or XTN_WRITE, address field must be one of these values, it determines the extended access type.

```
enum ExternalAccessType {
    XTN_PTPR,
    XTN_TLB_EN,
    XTN_ICACHE_FLUSH,
    XTN_DCACHE_FLUSH,
    XTN_ITLB_INVAL,
    XTN_DTLB_INVAL,
    XTN_ICACHE_INVAL,
    XTN_DCACHE_INVAL,
    XTN_ICACHE_PREFETCH,
    XTN_DCACHE_PREFETCH,
    XTN_SYNC,
    XTN_INS_ERROR_TYPE,
    XTN_DATA_ERROR_TYPE,
    XTN_INS_BAD_VADDR,
    XTN_DATA_BAD_VADDR,
};
```

Instruction request and response

Instruction request, only significant if `valid' is asserted. addr must be 4-byte aligned.

```
struct InstructionRequest {
    bool valid;
    addr_t addr;
    enum ExecMode mode;
};
```

Instruction response.

Valid is asserted when query has beed satisfied, if no request is pending, valid is not asserted.

instruction is only valid if no error is signaled.

```
struct InstructionResponse {
    bool valid;
    bool error;
    data_t instruction;
};
```

Data request and response

Data request, only significant if `valid' is asserted. addr must be 4-byte aligned. wdata is only significant for be-masked bytes.

- wdata[7:0] is at ![addr], masked by be[0]
- wdata[15:8] is at [addr+1], masked by be[1]
- wdata[23:16] is at [addr+2], masked by be[2]
- wdata[31:24] is at [addr+3], masked by be[3]

When type is XTN_READ or XTN_WRITE, addr must be an opcod of enum ExternalAccessType. For extended access types needing an address, address is passed through the wdata field.

```
struct DataRequest {
    bool valid;
    addr_t addr;
    data_t wdata;
    enum DataOperationType type;
    be_t be;
    enum ExecMode mode;
};
```

Data response.

Valid is asserted when query has beed satisfied, if no request is pending, valid is not asserted.

data is only valid if no error is signaled.

Read data is aligned with the same semantics than the wdata field in struct DataRequest. Only bytes asserted in the BE field upon request are meaningful, others have an undefined value, they may be non-zero.

```
struct DataResponse {
    bool valid;
    bool error;
    data_t rdata;
};
```

Functions

void reset()

Reset operation, Iss must behave like the processor receiving a reset cycle.

Tell the Iss to execute *at most* ncycle cycles, knowing the value of all the irq lines. Each irq is a bit in the irq_bit_field word.

uint32_t executeNCycles(uint32_t ncycle, uint32_t irq_bit_field)

Iss must return the number of cycles it actually executed. This is at least 1, at most ncycle.

void getInstructionRequest(struct InstructionRequest &)

Iss must populate the request fields.

void setInstruction(const struct InstructionResponse &)

Iss is given back the response. It may not be valid.

void getDataRequest(struct DataRequest &)

Iss must populate the request fields.

void setData(const struct DataResponse &)

Iss is given back the response. It may not be valid.

void setWriteBerr()

The cache received an imprecise write error condition, this signalling is asynchronous.

Other APIs

Sideband signals

In order to inform the ISS about some cache caracteristics, those functions have been defined.

Their implementation is optional.

void setICacheInfo(size_t line_size, size_t assoc, size_t n_lines)

Inform the Iss about the instruction cache caracteristics

void setDCacheInfo(size_t line_size, size_t assoc, size_t n_lines)

Inform the Iss about the data cache caracteristics

Debugger API

This API is optional, it serves to expose the internal ISS registers to a debugger.

The debugger API is ISS-architecture independant.

unsigned int debugGetRegisterCount()

Iss must return the count of registers known to GDB. This must follow GDB protocol for this architecture.

debug_register_t debugGetRegisterValue(unsigned int reg)

Accessor for an Iss register, register number meaning is defined in GDB protocol for this architecture.

void debugSetRegisterValue(unsigned int reg, debug_register_t value)

Accessor for an Iss register, register number meaning is defined in GDB protocol for this architecture.

size_t debugGetRegisterSize(unsigned int reg)

Get the size for a given register. This is defined in GDB protocol for this architecture.

addr_t debugGetPC()

Get the current PC

void debugSetPC(addr_t)

Set the current PC